603 414

A FACILITY AND ARCHITECTURE FOR AUTONOMY RESEARCH

Greg Pisanich
QSS Group Inc., NASA Ames Research Center
Moffett Field, CA

Abstract

Autonomy is a key enabling factor in the advancement of the remote robotic exploration. There is currently a large gap between autonomy software at the research level and software that is ready for insertion into near-term space missions. The Mission Simulation Facility (MSF) will bridge this gap by providing a simulation framework and suite of simulation tools to support research in autonomy for remote exploration. This system will allow developers of autonomy software to test their models in a high-fidelity simulation and evaluate their system's performance against a set of integrated, standardized simulations.

The Mission Simulation ToolKit (MST) uses a distributed architecture with a communication layer that is built on top of the standardized High Level Architecture (HLA). This architecture enables the use of existing high fidelity models, allows mixing simulation components from various computing platforms and enforces the use of a standardized high-level interface among components. The components needed to achieve a realistic simulation can be grouped into four categories: environment generation (terrain, environmental features), robotic platform behavior (robot dynamics), instrument models (camera/spectrometer/etc.), and data analysis. The MST will provide basic components in these areas but allows users to plug-in easily any refined model by means of a communication protocol. Finally, a description file defines the robot and environment parameters for easy configuration and ensures that all the simulation models share the same information.

Biography

Greg Pisanich is a Technical Area Liaison for the QSS Group Inc. within NASA Ames Research Center's Code IC and is Project Manager of the Mission Simulation Facility. He holds Master's degrees from Embry Riddle Aeronautical University and Santa Clara University. His background and interests include aviation, unmanned aviation systems (UAVs), simulation, robotics, autonomy, cognitive modeling, and human factors.

A FACILITY AND ARCHITECTURE FOR AUTONOMY RESEARCH

Greg Pisanich
QSS Group Inc., NASA Ames Research Center [gp@email.arc.nasa.gov]
Moffett Field, CA

INTRODUCTION

Autonomy is a key enabling factor in the advancement of the remote robotic exploration. Such systems will need to demonstrate high levels of autonomy and adaptability to accomplish their tasks without continuous human control or intervention (Washington et. al, 1999).

In order to accelerate the development of this technology, NASA initiated the Intelligent Systems (IS) program managed out of the Ames Research Center. In addition to sponsoring research in autonomous systems and algorithms, they also had the foresight to initiate the development of simulation resources, tools and facilities.

Commercial software tools developed for industrial robot simulation have been available for years, but autonomous systems researchers have been unable to take advantage of these tools because they are not flexible enough to be able to represent newly designed robotic systems evolving in unstructured environments.
The result has been that each research lab has had to develop its own simulator. Because of the complexity and time required for their development, such simulators have usually been oriented towards "block-world" models (answering most needs while keeping simplicity). These tools are useful for experimenting with robotic autonomy, but are not suitable for planetary missions due to their 2D world model and limited sensors.

At the same time, sophisticated simulators have been developed within research laboratories to address specific problems such as robot dynamics (Yen, Jain, & Balaram, 1994) or instrument and environment modeling (Thompkins, 1999). These tools offer highly accurate models but are oriented towards engineering design or mission-ready simulation. Consequently, even though many high fidelity models exist, they are difficult to combine into an integrated simulation. Typically these tools are tied to a specific operating system or computer language and are not designed for applications outside of their nominal scope.
The Mission Simulation Facility (MSF) project was initiated to bridge this gap. This will be achieved through the development of the Mission Simulation Toolkit (MST), a software package comprising 1) a framework for connecting and synchronizing distributed software models, 2) generic interfaces abstracted from the transport layer, and 3) a set of basic components required for a simulation.

DEVELOPMENT GOALS

We have established several goals that the Mission Simulation Toolkit must achieve:

- Provide a software framework for the development of differing levels of autonomy.
- Allow easy integration of autonomy modules and tools into the simulation.
- Be easily extensible to multiple robotic platforms and environments.
- Allow interchangeability of real hardware and simulated components.
- Be easily distributable to external groups (NASA Sites, Educational, and Research Facilities).
- Provide varying levels of simulation fidelity.

SCOPE AND APPLICATIONS

The MST architecture has been designed to support multiple mission platforms (e.g. planetary robots, spacecraft, underwater vehicles), with the initial focus on planetary rovers.

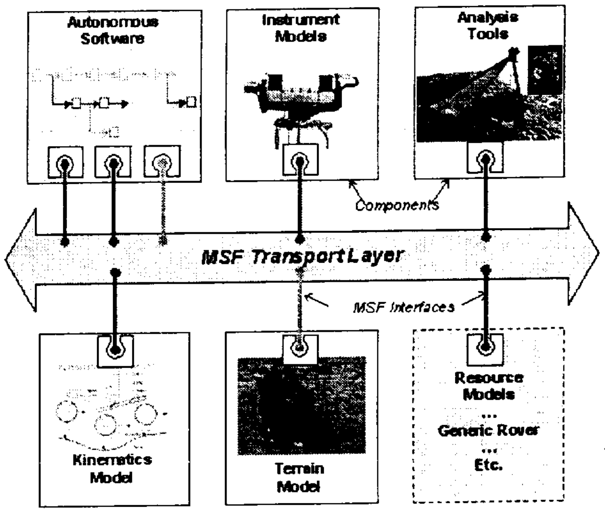

Figure 1. Overview of the Mission Simulation Toolkit

The components needed for a rover simulation would include a terrain model, site information (such as coloration and mineral composition), an environment model (sun position, lighting, temperature, etc), and a rover including a kinematic model and on-board sensors, and several scientific instruments. It is up to the user to decide what granularity of models best suits the purpose of his simulation.

For example, a user who is concerned primarily with collecting scientific data may not require a sophisticated rover model because he may not care how the rover gets from one point of interest to another. On the other hand, a user who is developing a trajectory generator may want to control the individual wheels of a rover. For this reason the MST provides both high and low level interfaces to a number of standard models.

This dual level interface is also needed to support a variety of robot architectures. Since very different approaches currently exist, see (Coste-Maniere & Simmons, 2001) for examples, the MST must provide an interface generic enough to equally support hierarchical, behavioral or hybrid robot architectures. Figure 1 shows the concepts of the MST distributed simulation relying on a common communication framework to connect models and autonomy software.

DISTRIBUTED ARCHITECTURE

The MST architecture is derived from two main requirements: to support distributed simulation on multiple platforms and to ensure extensibility through an open architecture.

Multiple Platform Support

Users of the MST (autonomy developers) typically develop their tools in a variety of environments, for example, a LISP program under Solaris on a Sun workstation or a C++ program under Linux on an Intel PC. Target systems for the autonomy software (the rover control software) may also be developed on different operating systems and hardware platforms. Such software could for instance rely on a particular flavor of Linux running on a PC-104 Pentium board, or could be a dedicated embedded system running VxWorks. The MSF project does not intend to develop all the simulation components but rather will take advantage of existing tools. To minimize the adaptation requirements, each particular software component should be usable by the MST on the original platform for which it was developed.

Open Architecture

The MST is a general-purpose testbed for mission simulation rather than a specific simulator, which implies that different sets of components will be used for different scenarios or different domains. For instance, one might use a kinematics model for a rover and a fluid dynamics model for an underwater vehicle. In addition, a component should be usable in multiple scenarios, which means that the same rover kinematics model could be used for various rovers with particular payloads operating in different environments. Finally, it should be possible to replace a component of a specific type with another that performs the same function but at a different level of fidelity. A simple kinematics rover simulator could be adequate to test high-level autonomy concepts such as path planning, while a dynamics rover simulator including accurate soil-wheel interactions may be required to test an autonomous control system for the mobility of the rover. It is therefore essential that the MSF define clear interfaces between the components to facilitate the exchange of components included in the MST with those developed at other research institutions.

A way to satisfy the above requirements is to have a distributed architecture where components communicate with each other using a common transport layer. The MST is built on top of the standardized High Level Architecture (HLA), which is an architecture for simulation reuse and inter-operability developed by the Defense Modeling and Simulation Office (DMSO). The MST currently uses the Runtime Infrastructure (RTI), a software implementation of HLA (Kuhl, Weatherly, & Dahmann, 2002) freely distributed by DMSO. The HLA/RTI provides the MST the following services:

- Multi-platform support: IRIX, Solaris, Linux, Win32 and VxWorks
- C++ and Java bindings
- Choice of transport protocol: TCP (reliable) or UDP (fast)
- Publish/Subscribe scheme
- Communication through objects or messages
- Various time management schemes for simulation synchronization

To facilitate the integration of components in an MST-based simulation, an abstraction layer has been developed on top of the HLA, the Federate ToolKit (FTK). FTK is responsible for the integration of communication entities with the Runtime Infrastructure. The communication objects and messages defining the MST interface are easily designed using the Unified Modeling Language (UML). All the necessary C++ code to use these communication entities is generated automatically.

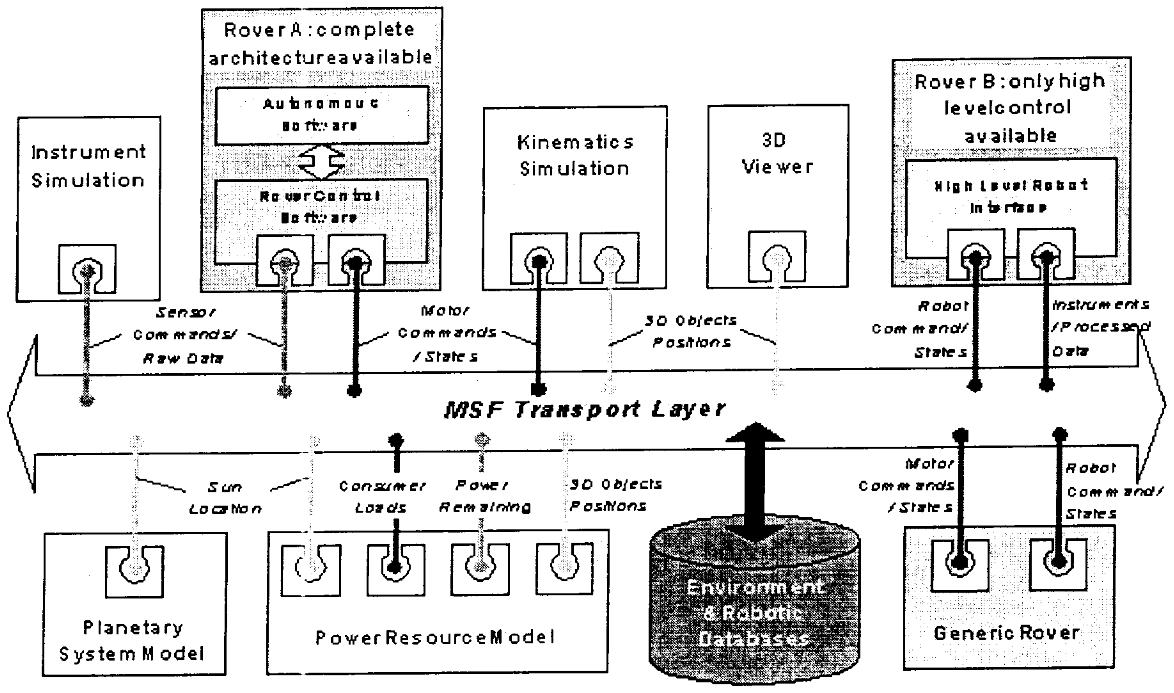

Figure 2: MST simulation showing several components communicating through common interfaces.

Figure 2 provides an example of an MST-based simulation with several interconnected components. Two separate rover autonomy software executions are participating in the simulation: The software in Rover A is provided by a lab having a complete rover architecture (and probably a real rover) from the high level control down to the hardware control; the software on Rover B comes from a lab working only on high level autonomous algorithms and without hardware control. When the Rover A software sends commands to its actuators (e.g. motor1_start(speed, duration)), the commands are routed to the simulator rather than to real hardware. The kinematics simulator accepts such commands and computes the behavior of the rover on the terrain regarding these inputs. When Rover B issues a high level command to its base controller (e.g. roverB_moveto (position, obstacle avoidance=on)), the Generic Rover model catches this message and produces motor commands for a simple model of a rover, causing it to move from one location to another while avoiding obstacles. These motor commands can be processed by the same kinematics simulator used to control Rover A. The same scheme is used when controlling instruments, generalized as sensors in Figure 2 (the figure does not show the full path of information flow). In addition to the Kinematics and Instruments models, a Power Resource model is participating in the simulation. It monitors the load of each actuator as well as the power generated by solar panels and computes the power remaining in the rover's batteries. The power output of the solar panels depends on the orientation of the solar panels relative to the sun. The kinematics model provides the position of the panels and the sun's position is delivered by another component computing the Ephemeredes. This example shows how different components are reusable for different scenarios, and how the definitions of the networked MST interfaces removes all the dependencies between the components.

THE MISSION SIMULATION TOOLKIT

A first prototype of the Mission Simulation ToolKit was demonstrated in June 2002. It is being evaluated by autonomy researchers within NASA Ames and is being applied to the development of contingent plan execution. Figure 3. shows the components and messages that make up this release. The following paragraphs describe the software that are included in this internal release :

Federate Tool Kit (FTK)

The federate tool kit is a set of C++ classes forming a layer above HLA. The FTK classes simplify much of the development overhead of HLA, which includes

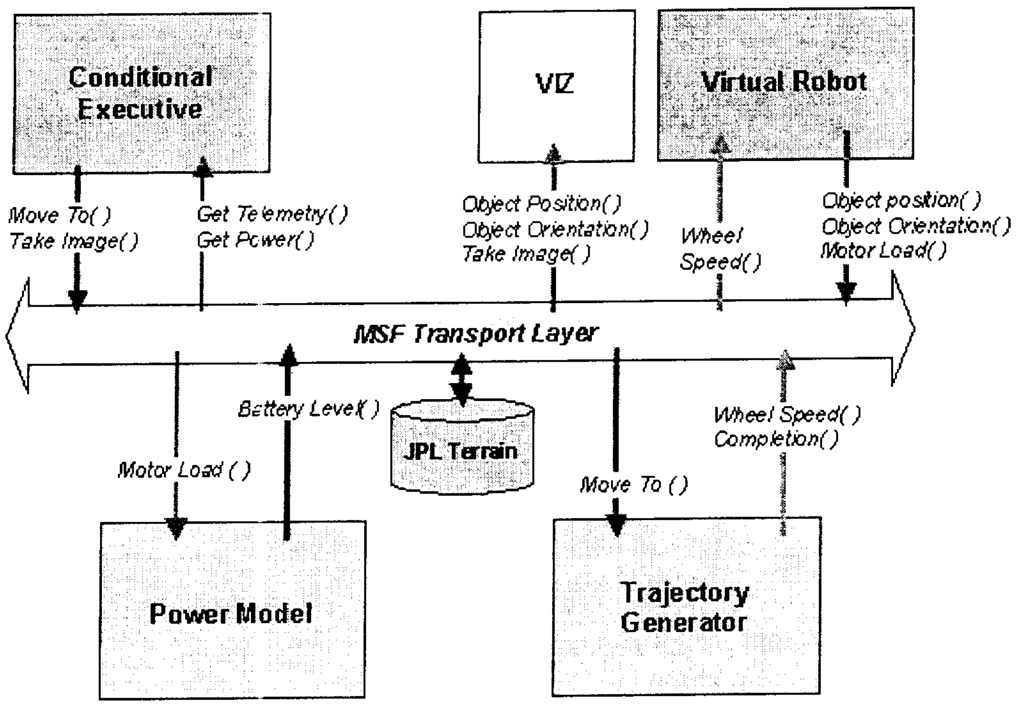

Figure 3. Components and messages of the June 2002 MST release.

tasks such as joining and exiting, and managing attribute updates and attributes. The FTK also maintains a local representation of the HLA simulation for each component. The FTK objects have also been developed enforce proper behavior of components in the simulation: They abstract the developer from operations and sequences in HLA that will ensure proper interaction with other federation objects and the federation as a whole.

UML2HLA

UML2HLA has been developed to map UML communications entities to an HLA implementation. UML2HLA has been implemented in the form of a Visual Basic plugin to Rational Rose that inputs a MST Communication Objects Hierarchy and provides two outputs. The first is an HLA FED file that is used by HLA to describe the allowable objects in the simulation Federation. The second is a C++ class library that can be used by a simulation developer to access those objects.

These generalized classes are based on the FTK layer, allowing the communications objects to inherit the facilities for accessing the HLA transport layer provided by FTK.

MST Communication Object Hierarchy

The Communications Hierarchy is described in the form of UML (Unified Modeling Language) files. The Communication Object Hierarchy describes a set of objects that will exist in the simulation, their attributes, and the communication messages that can be sent or received by each object.

The Object Hierarchy is easily modified and expanded to support additional objects or messages using any UML tool. The Object Hierarchy as currently designed is specialized for the simulation of Planetary Rovers, however it can and will be expanded to support other domains.

The conceptual communications objects are implemented as a set of C++ classes using the UML2HLA Process.

VIZ

The inclusion of the VIZ program module allows the visualization of MST simulations. VIZ supports the visualization of the terrain, the robot(s), and the output of sensors such as cameras. VIZ also allows the simulation developer to assess the progress and operation of the robot using visual tools.

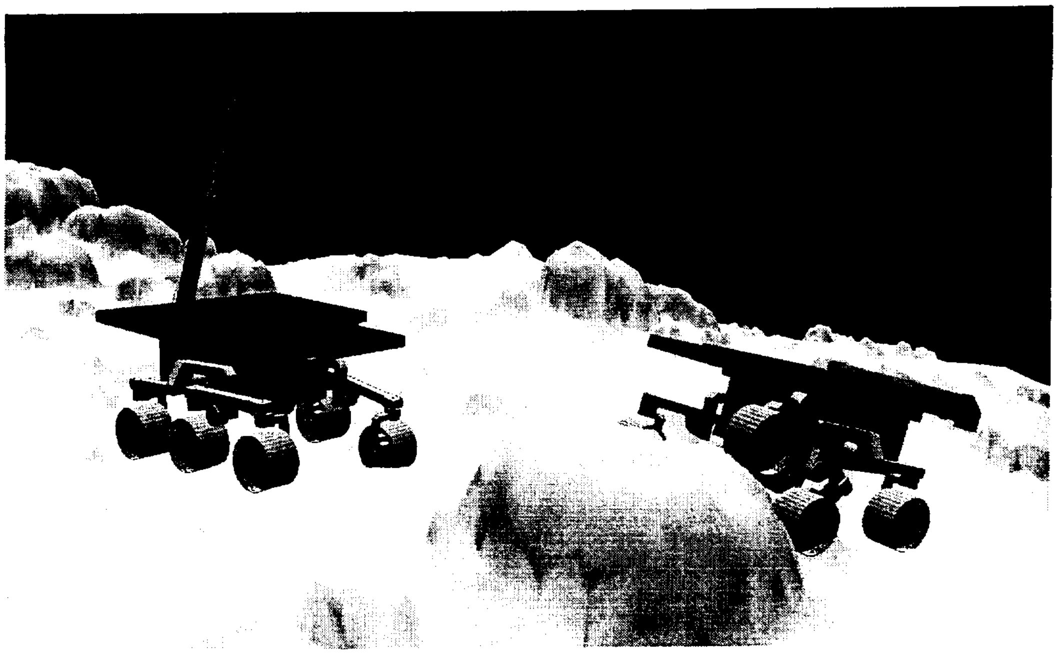

Figure 4. An MST simulati n involving two rovers on simulated Martian terrain as visualized in VIZ.

The VIZ software was developed at NASA Ames. The MST team has developed an HLA interface to VIZ that allows it to interact with MST simulation components. Figure 4 shows example output of the VIZ tool.

Terrain

The MST provides the capability of introducing terrain models (consisting of DEM and albido) into the simulation. The MST supplies several models that were synthesized and generated by the JPL Super Computing Group based on Martian data models. The MST also maintains the capability of introducing and using terrain data from measured (real) models.

Models

The MST currently provides models of systems and processes that are representative of planetary robotic systems. These include a 3D robot model, power model, kinematic model, and robot movement generator. The MST will be expanded to include additional models, including sensors and instruments (cameras, spectrometers), effectors (arms, booms) and other subsystems.

USER GROUPS

We are developing the MST to support two different user groups. Our use of a component architecture should allow us to support both with minimal changes.

The first group involves primarily internal NASA (Ames and JPL) researchers, including those that are involved in the IS Program. This group is distinguished by their need to use components that may be export controlled or otherwise cannot be released outside of NASA. These users would gain access to these components from internal sources.

The second group includes users primarily external to NASA, which would include educational and other research institutions, including foreign groups. These users would have access to a full range of components that will have been developed with distribution in mind. These components may have been developed internally or be contributed by users of the MST. This group also includes foreign students and researchers that may be working within NASA. These users would be using the MST to develop algorithms that don't involve export-controlled software.

RELEASE STRATEGY

The release of the MST will involve several distributions and capabilities over a nominal 3-year period. The releases should be scheduled to correspond to the levels of capabilities and refinements to the MST architecture and code.

The initial release of the MST will be to internal IS researchers (NASA Ames and JPL) and no more than three university groups. This release will be used as a test of the robustness of the code and the distribution process. We expect to learn a lot about how well we have documented the release and what things are missing or need to be added to the MST to help the user. This release should be via a CD is scheduled for Winter 2002.

The second release will be to internal users and will be capable of linking in export-controlled components. This release is scheduled for Summer 2003. We will evaluate the use of the web for distribution of the software.

The third major release of the MST will include additional components and will be targeted at a larger group of external users. These should include both US and foreign universities and research groups. We expect that this version may involve the sharing and exchange of component modules between groups. We expect that this release will be done in full via the web. It is currently scheduled for Winter 2003.

During 2004, we will work to develop an open source version of the MST, which may involve working with an external group that focuses on this process. We expect that this version may also be licensable by commercial companies via the NASA Commercial Technology Office. We also expect that this version may allow users to download the MST core architecture from an open source site and download other modules from the sites of other MST affiliates. This release is nominally scheduled for Winter 2004.

SUMMARY

There is currently a large gap between autonomy software at the research level and software that is ready for mission insertion. The Mission Simulation Facility will bridge this gap by providing a simulation framework and suite of simulation tools (the Mission Simulation ToolKit) to support research in autonomy for remote exploration. This system will allow developers of autonomy software to test their models in a high-fidelity simulation and evaluate their system's performance against a set of integrated, standardized simulations.

ACKNOWLEDGEMENTS

This project is funded under the NASA Intelligent Systems (IS) Program, which is managed by Butler Hine.

The author acknowledges the hard work and many contributions of the MSF development team: Lorenzo Flueckiger, Christian Neukom, Michael Wagner, and Laura Plice.

The author would also like to acknowledge the use of software, development contributions, and feedback provided by Rich Washington (Conditional Executive) and Larry Young (VIZ).

REFERENCES

Washington, R., Golden, K., Bresina, J., Smith, D. E., Anderson, C., and Smith, T. (1999). Autonomous rovers for Mars exploration. In *Proceedings of The 1999 IEEE Aerospace Conference.*

Tompkins, P., Stentz, A., and Whittaker, W.L. (2001). Automated surface mission planning considering terrain, shadows, resources and time. In *Proceedings of i-SAIRAS 2001.*

Yen, J., Jain, A., and Balaram, J. (1999). ROAMS: Rover Analysis, Modeling and Simulation. In *Proceedings of i-SAIRAS 1999*, pages 249–254.

Coste-Manière, E. & Simmons, R. (2001). Architecture, the backbone of robotic systems. In *Proceedings of the ICRA 2001.*

Kuhl, F., Weatherly, R., and Dahmann, J. (2000). *Creating Computer Simulation Systems: An Introduction to the High Level Architecture.* Prentice Hall.